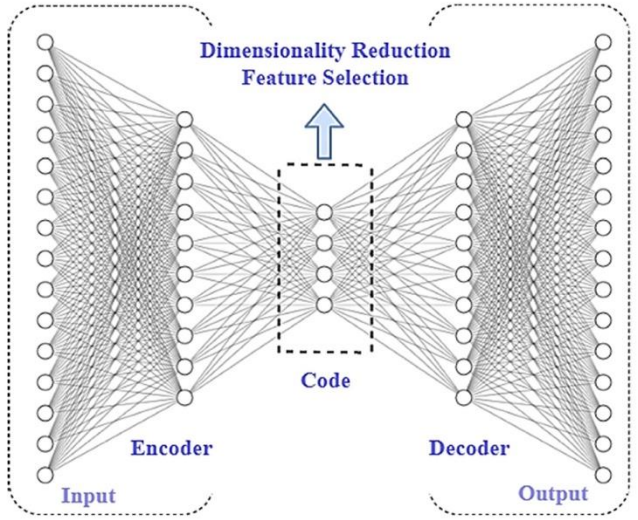
ASSA 2025

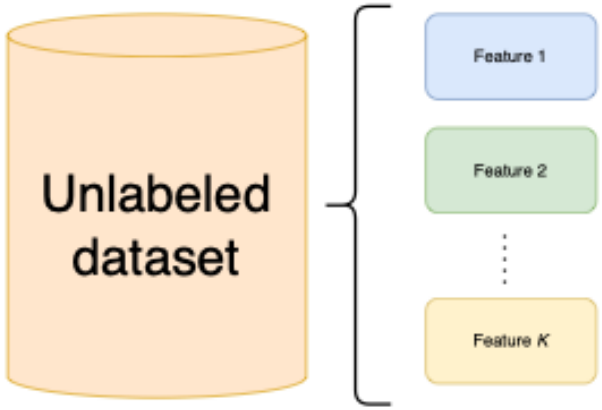
Machine Learning School – Day 2:

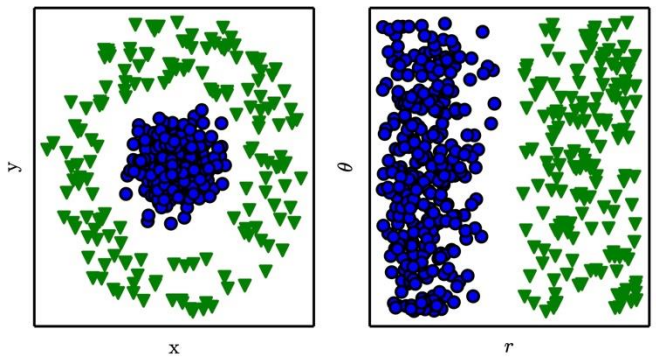
Unsupervised Learning

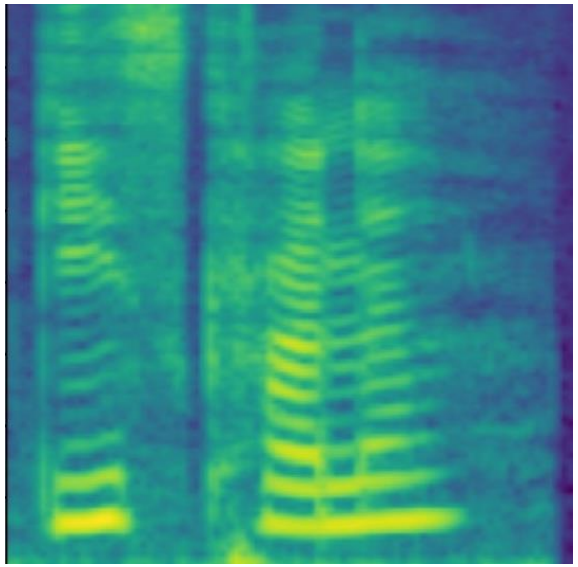
Elias Zea, KTH

Evren Yarman, SLB, KTH









Today’s schedule

Morning session:

Lecture 1. Representation-learning methods
Lecture 2. Clustering methods

Afternoon session:

Tutorial 1. Dictionary-learning assignment
Tutorial 2. Clustering assignment

Outline of lectures

Lecture 1. Representation-learning methods (Elias)

09:00 – 09:40: Unsupervised learning and linear approaches
09:50 – 10:30: Non-linear approaches and sparse coding

Lecture 2. Clustering methods (Evren)

10:50 – 11:30: Centroid-based techniques
11:40 – 12:20: Mixture models

Outline of lectures

Lecture 1. Representation-learning methods

09:00 – 09:40: Unsupervised learning and linear approaches

09:50 – 10:30: Non-linear approaches and sparse coding

Lecture 2. Clustering methods

10:50 – 11:30: Centroid-based techniques

11:40 – 12:20: Mixture models

What is Unsupervised Learning (UL)?

Supervised Learning Algorithms: Learn a mapping from input samples to output targets, commonly used for solving **classification** and **regression** problems.

$$data_{training} = \{input_i, label_i\} \Rightarrow Relationship$$

Unsupervised Learning Algorithms: Discover hidden patterns, structures, or groupings within data without relying on labeled outputs, commonly used for **clustering**, **dimensionality reduction**, and **representation learning** problems.

$$data_{training} = \{input_i\} \Rightarrow label_i$$

Association of these structures to patterns
↔ Choice of dictionary ↔ Domain knowledge

Why UL in acoustics?

- Labeling acoustic data is expensive
- Useful for exploratory analysis
- Applications:
  - **Seismic interpretation:** Clustering groups seismic traces with similar waveforms to identify geological structures.
  - **Environmental sounds:** Clustering identifies distinct vocalization patterns in whale songs among marine mammals.
  - **Speech processing:** Clustering segments audio by speaker to distinguish different voices in conversations.
  - **Music software:** Clustering frequencies and durations for reverse engineering musical pieces

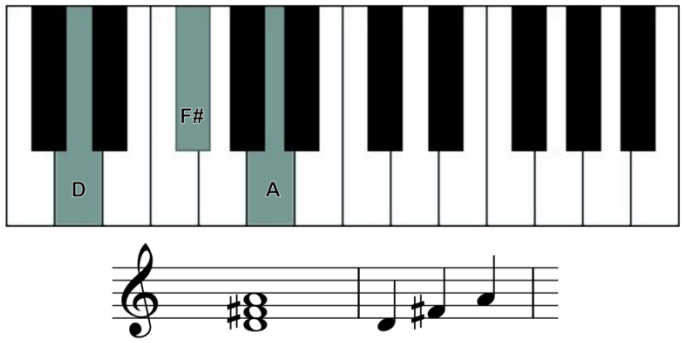
    Spotify by Daniel Ek and Martin Lorentzon.

    Auto-Tune by Andy Hildebrand, from Exxon Geologist to Autotune Inventor.

Waveform example

Examples of acoustic waveforms from music

$$f(t) = \mathrm{Re}\left\{\sum_m f_m e^{i\omega_m t}\right\}$$



SIAM Review
Vol. 44, No. 3, pp. 457–476

Time-Frequency Analysis of Musical Instruments*

Jeremy F. Alm†
James S. Walker‡

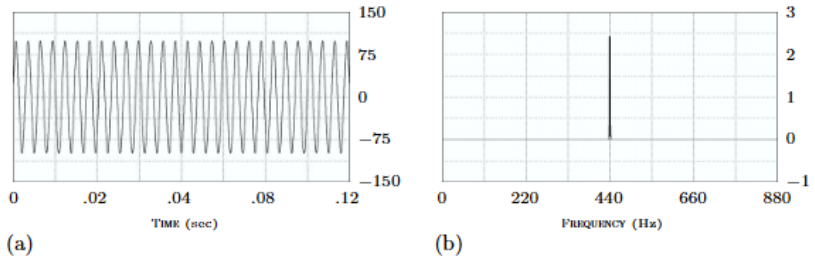


Fig. 2.1 *Fourier analysis of a pure tone.* (a) *Graph of a finite segment of a pure tone, 440 Hz.* (b) *Computer-calculated Fourier spectrum.*

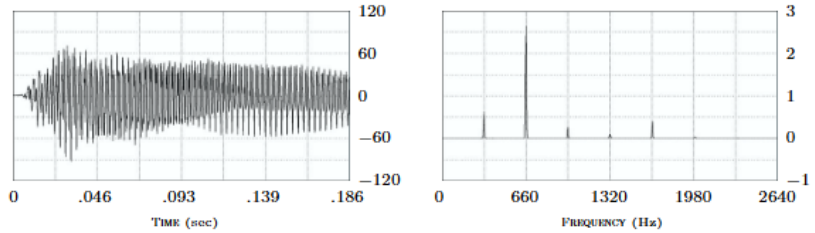


Fig. 2.2 *Fourier analysis of the piano note E_4 (E above middle C). (Note: The vertical scales of all spectra shown in this paper have been normalized to the same range.)*

Spectrogram example

- Time-frequency representation (Importance of choice of dictionary)

$$f(t) = Re\left\{\sum_{m,n} f_{m,n} e^{i\omega_{mn}t} w(t - t_n)\right\}$$

- Useful for audio classification
- Example: song analysis

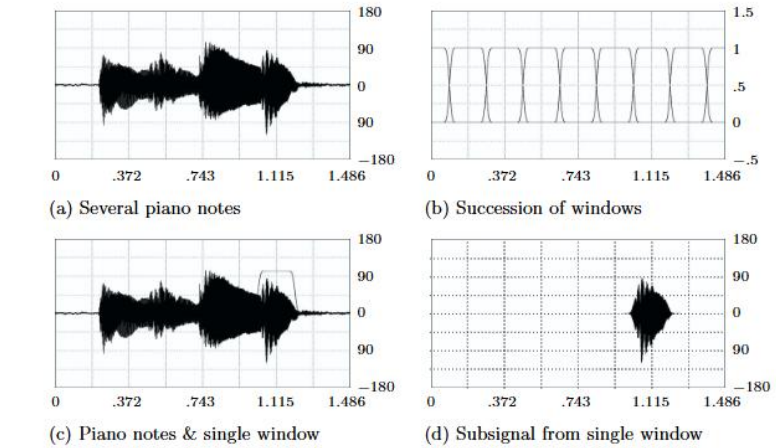
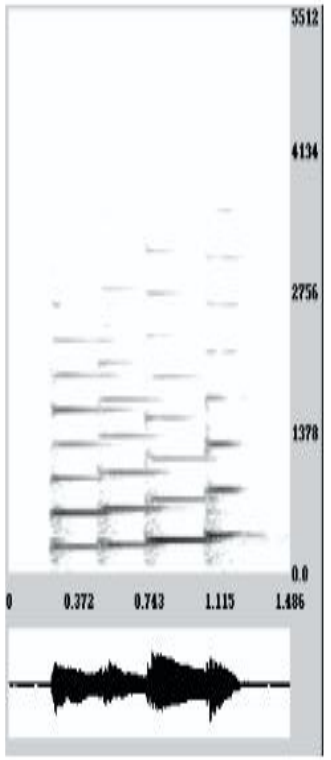


Fig. 3.2 *Components of a spectrogram.*



frequency

Time

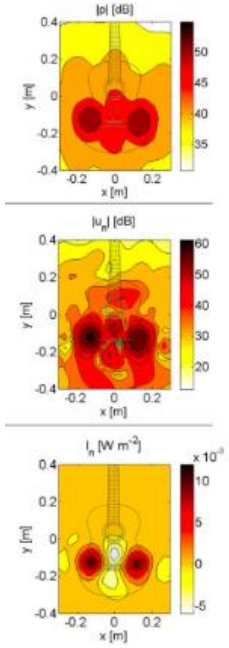
Flavors of UL

- Dimensionality reduction
- Representation learning
- Clustering
- Association rule learning
- ….

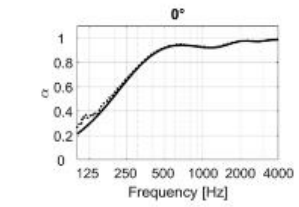
Dimensionality reduction in acoustics

Acoustic data often lives in high dimensions

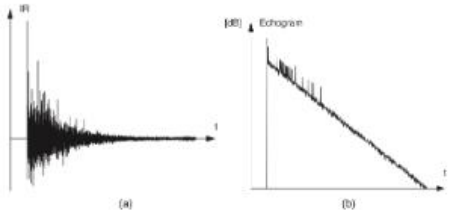
- Time samples
- Spatial samples
- Sound sources
- Boundary conditions
- Audio channels
- Bit rates
- …



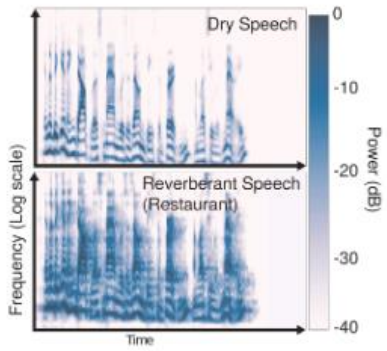
E. Fernandez-Grande *et al.* JASA **141**(1), 2017



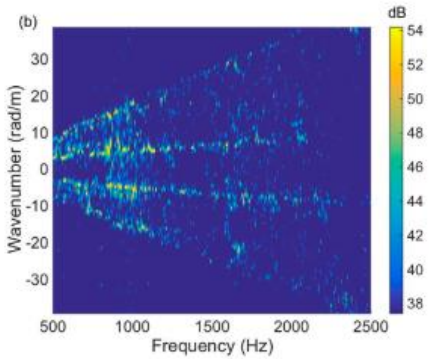
A. Richard *et al.* JASA **141**(6), 2018



L. Savioja & U.P. Svensson, JASA **138**(2), 2015



M. Bianco *et al.* JASA **146**(5), 2019

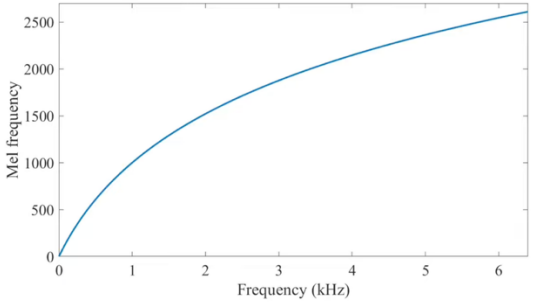


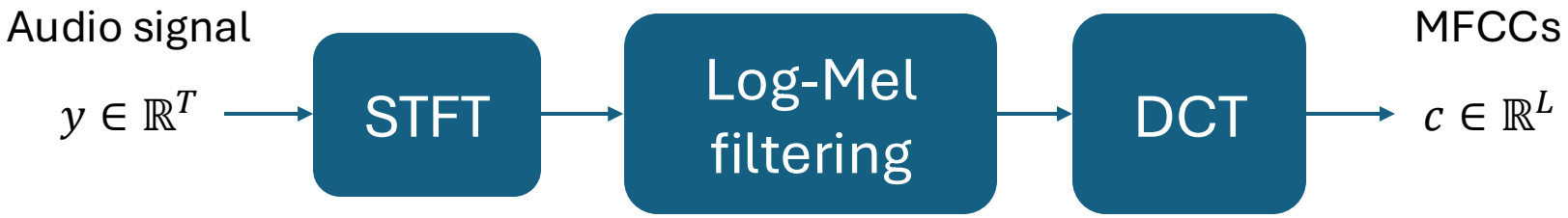
E. Zea *et al.* JSV **409**, 2017

Examples in time-frequency

Ex: MFCCs, mp3 format,…

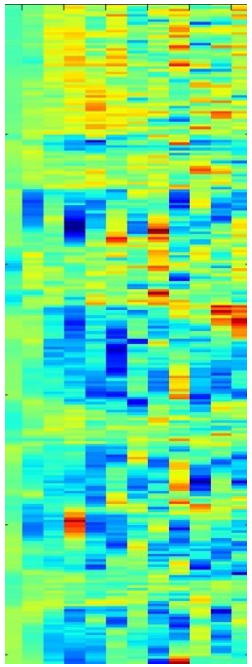
Goal: *Reduce* signal dimension informed by auditory system





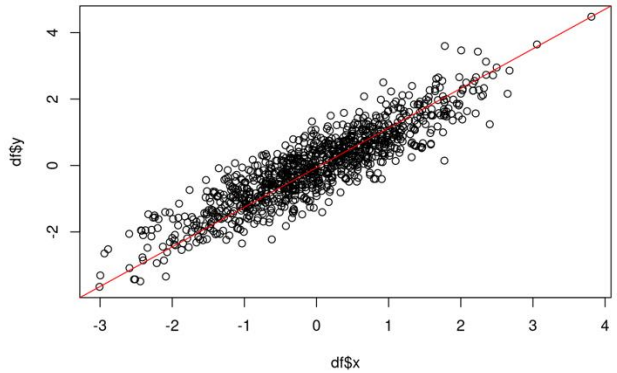
$T \gg L$

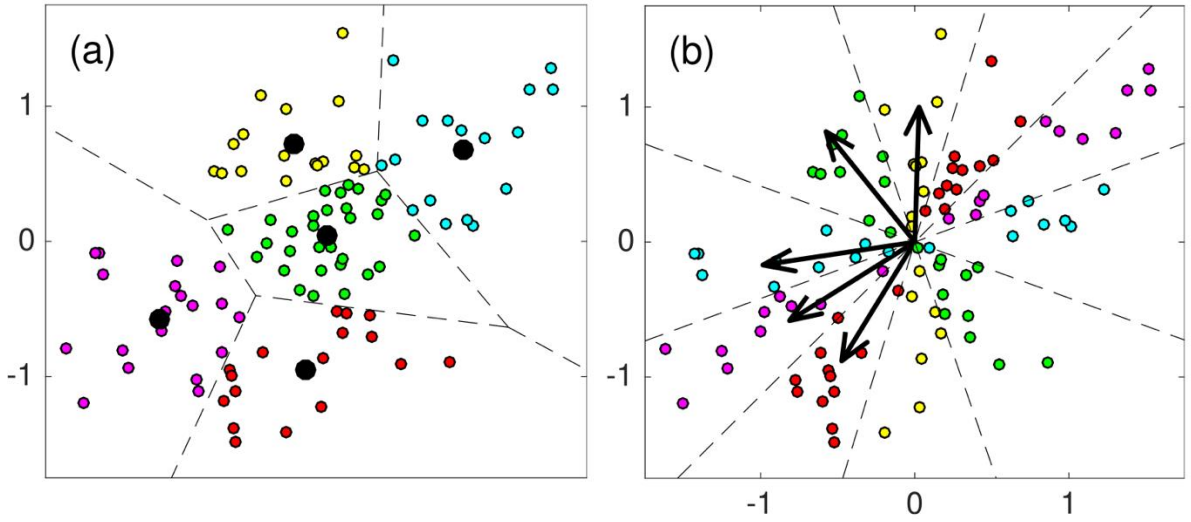
Dimensions of input vs dim of output

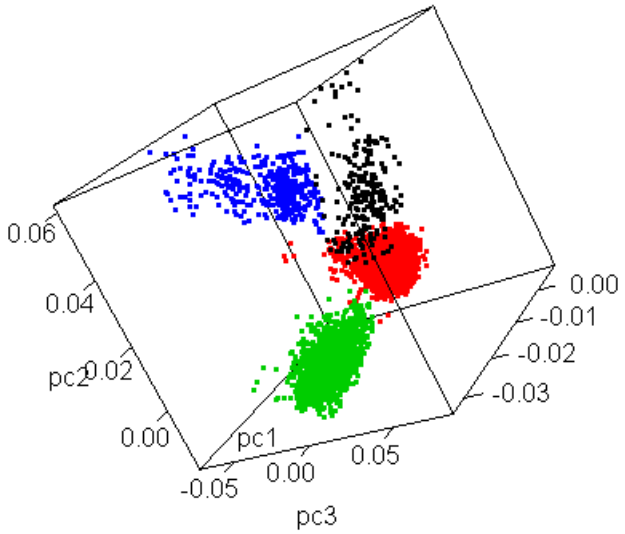


Our goal is then to…

Represent most of the variance in a dataset with lower-dimensional features







Usefulness principles

- Useful when raw data (features) are highly redundant or noisy, or for visualization or compression tasks.

- Less useful when the reduction leads to losing task-critical info.

Rule of thumb:

Do not reduce dimensionality below what's needed to capture the variability relevant to your task

Overview of common approaches

Linear feature projections

- PCA, SVD, LDA, ICA,…

Refs:

I.T. Jolliffe, Principal Component Analysis, 2nd Ed., Springer New York, NY (2002)
Murphy, Probabilistic Machine Learning: An Introduction, MIT Press, Cambridge, MA (2022)

Non-linear manifold learning

- Kernel-PCA, t-SNE, LLE,…

t-distributed stochastic neighbor embedding

Refs:

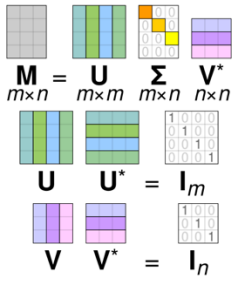
Wang *et al.*, J. Mach. Learn. Res. 22, 1–73 (2021)
McInnes *et al.*, J. Open Softw. 3, 861 (2018)
Amid *et al.*, arxiv:1910.00204 (2019)

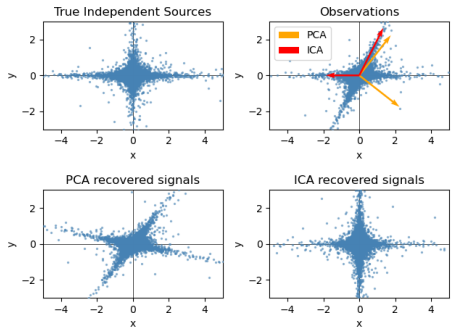
Neural network embeddings

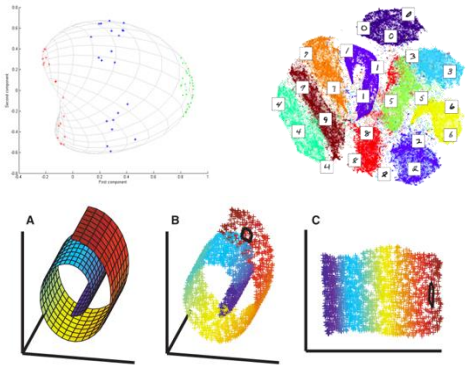
- AEs, VAEs, …

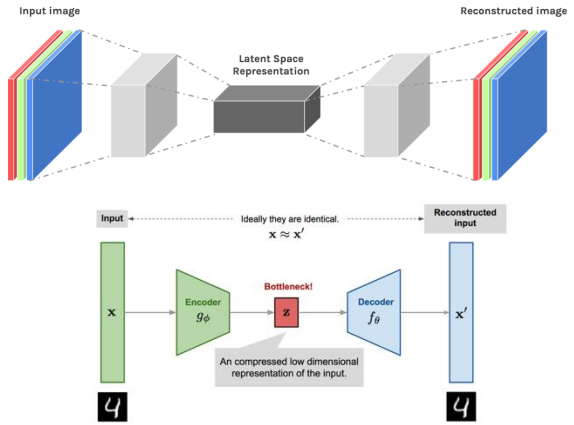
Refs:

Linhardt & Gupta, OCEANS 2022, 1–4 (2022)
Chien *et al.*, Computers Geotech. 155, 105233 (2023)
Gibb *et al.*, Ecol. Inform. 80, 102449 (2024)







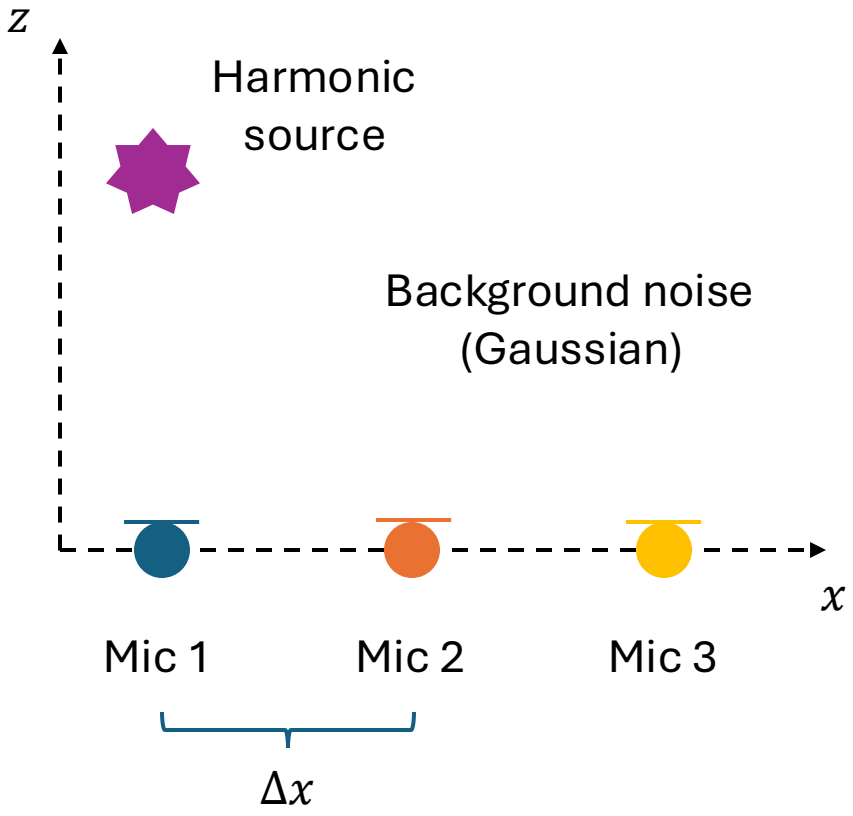


In this lecture, we will focus on…

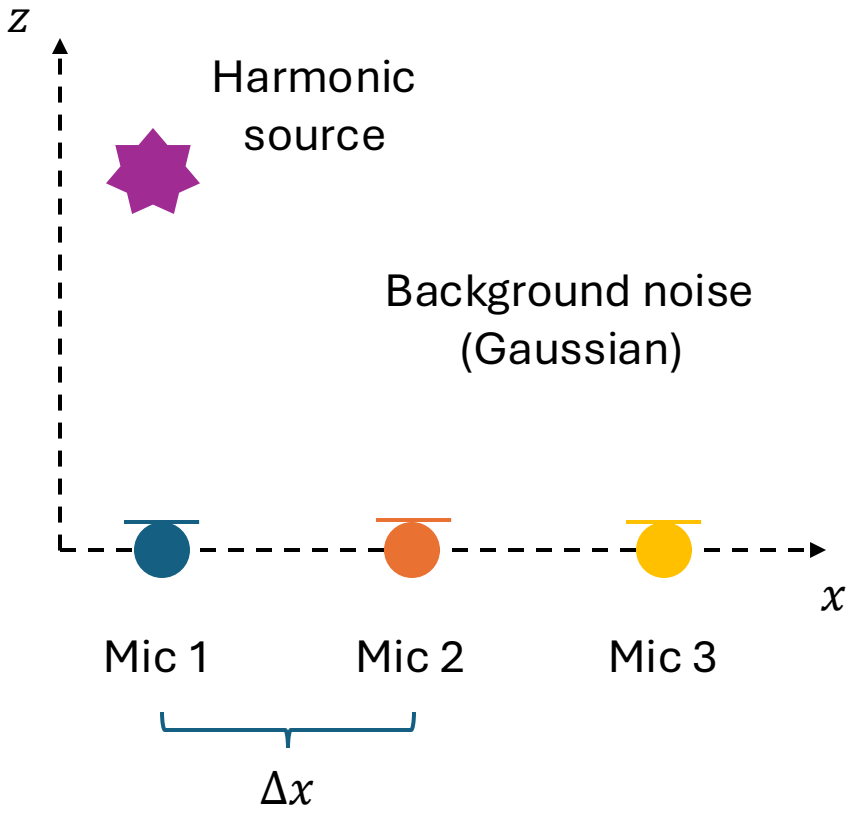
Going the linear way — Principal Component Analysis (PCA)

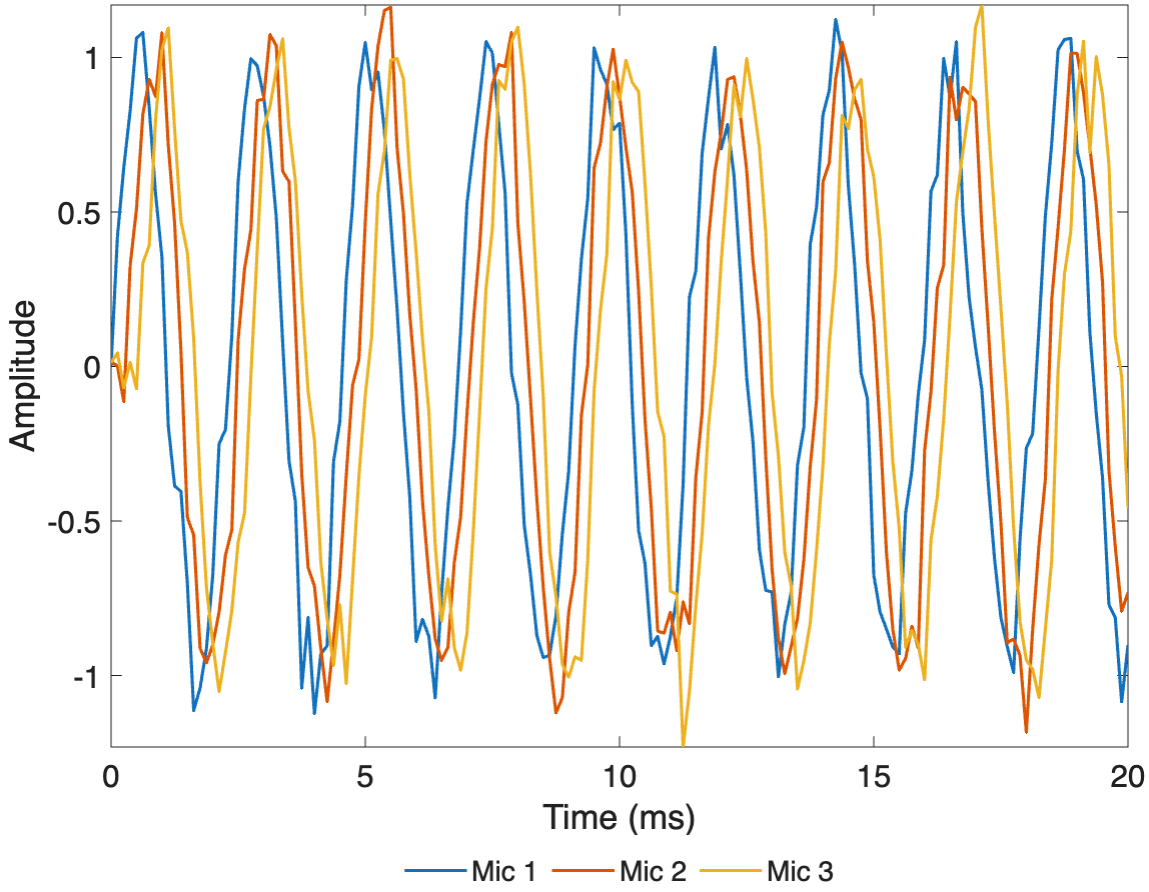
And beyond the linear way — Autoencoders (AEs), Sparse coding

The linear way – An example



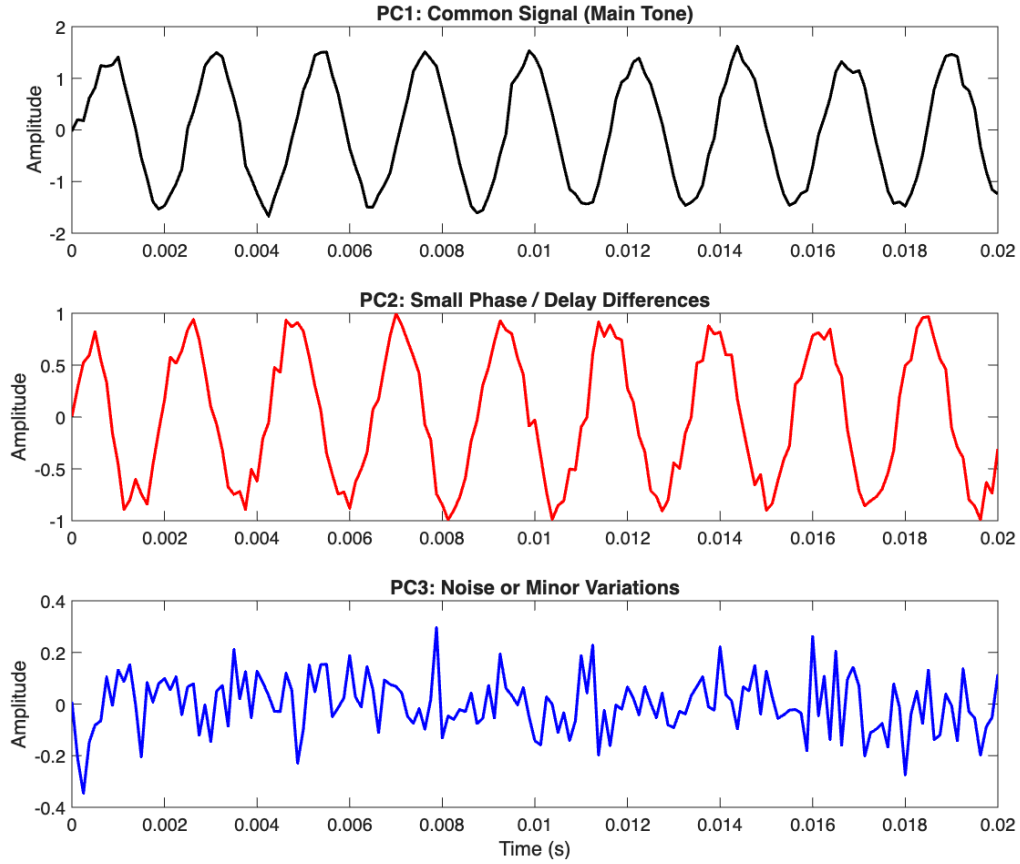
The linear way – An example





The linear way – An example

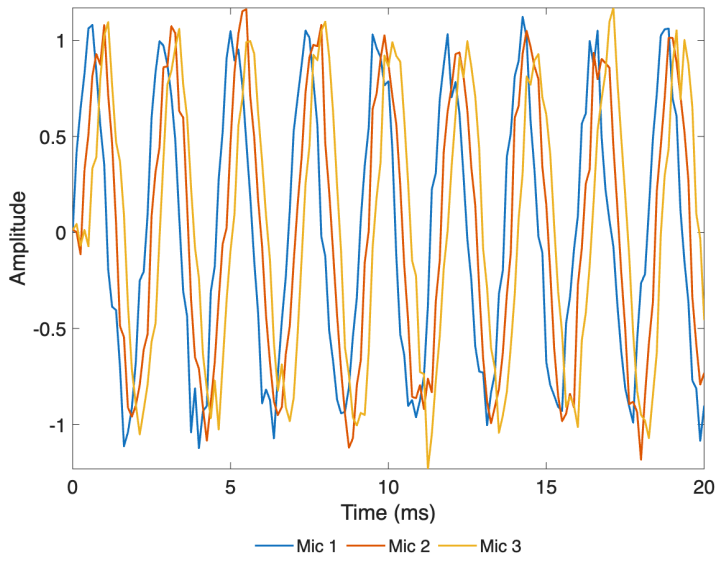
Rank determines the number of components



The linear way – An example

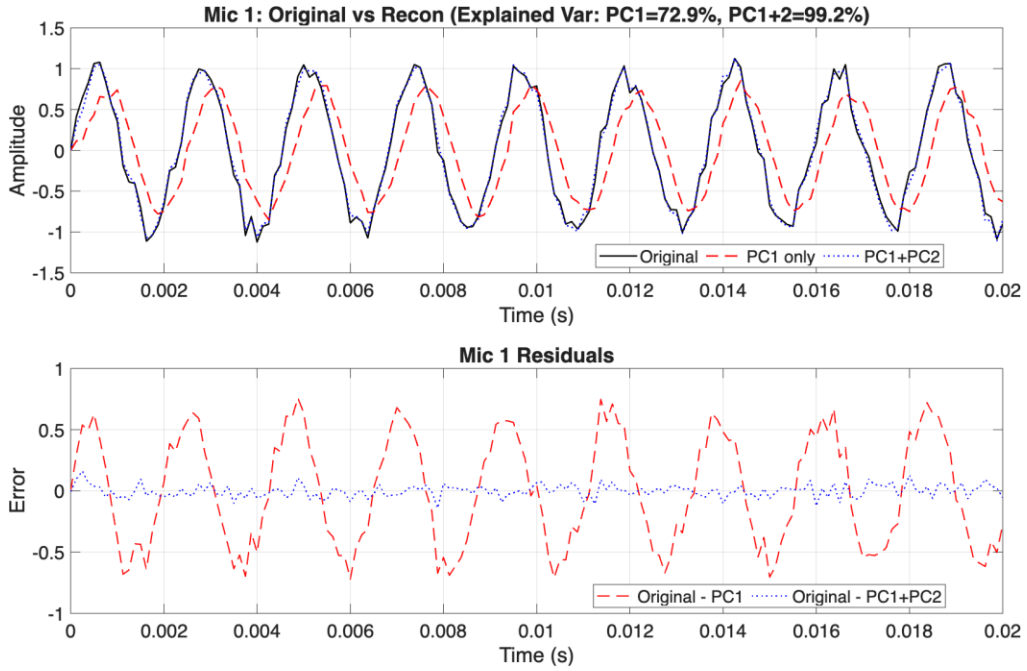
Dimensionality reduction illustrated (+ denoising)

3 features × 161 time samples



PCA

2 PCs x 161 time samples



The linear way – PCA

PCA in a nutshell:

1. Organize data: Rows → features; Columns → observations, zero-mean)

2. Learn orthogonal components from data (via SVD of covariance matrix)

3. Project data on the top K components to reduce dimensionality

Classic PCA textbook: I.T. Jolliffe, *Principal Component Analysis*, 2nd Ed., Springer New York, NY (2002).

The linear way – PCA

1. Consider a data matrix $\mathbf{X} \in \mathbb{R}^{N \times P}$, with zero-mean columns of N-dimensional vectors and P observations of such vectors (e.g., microphone positions in our example).

The linear way – PCA

1. Consider a data matrix $\mathbf{X} \in \mathbb{R}^{N \times P}$, with zero-mean columns of N-dimensional vectors and P observations of such vectors (e.g., microphone positions in our example).

2. Then, the PCs are the columns of (left-singular vectors in)

   $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_N] \in \mathbb{R}^{N \times r}$, with $r = \mathrm{rank}(\mathbf{X}) = \min(N, P)$

   of the covariance matrix $\mathbf{X}\mathbf{X}^{\mathrm{T}} = \mathbf{U}\mathbf{\Sigma}^2\mathbf{V}^{\mathrm{T}}$ (via SVD), with variances encoded in the singular values

   $\mathbf{\Sigma} = \mathrm{diag}(\sigma_1, \dots, \sigma_N)$, with $\sigma_1 \geq, \dots, \geq \sigma_N$.

The linear way – PCA

1. Consider a data matrix $\mathbf{X} \in \mathbb{R}^{N \times P}$, with zero-mean columns of N-dimensional vectors and P observations of such vectors (e.g., microphone positions in our example).

2. Then, the PCs are the columns of (left-singular vectors in)

   $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_N] \in \mathbb{R}^{N \times r}$, with $r = \mathrm{rank}(\mathbf{X}) = \min(N, P)$

   of the covariance matrix $\mathbf{X}\mathbf{X}^{\mathrm{T}} = \mathbf{U}\boldsymbol{\Sigma}^2\mathbf{V}^{\mathrm{T}}$ (via SVD), with variances encoded in the singular values

   $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1, \dots, \sigma_N)$, with $\sigma_1 \geq, \dots, \geq \sigma_N$.

3. Finally, the lower-dimensional projection (scores) of $\mathbf{X}$ is obtained via $\mathbf{B} = \mathbf{Q}^{\mathrm{T}}\mathbf{X} \in \mathbb{R}^{P \times K}$, with $\mathbf{Q} = \mathbf{U}_{[:,1:K]} \in \mathbb{R}^{N \times K}$ the first K PCs.

The linear way – PCA

When is it useful?

Optimal for linear Gaussian data and most effective dimensionality reduction when factors describing the data variance can be disentangled into orthogonal features.

Fast computations, often a baseline for more complex tasks.

Computational complexity: $\mathcal{O}(P^2 \cdot N + P^3)$

PCA – Frameworks

Some relevant frameworks

1. scikit-learn

```python
import numpy as np
from sklearn.decomposition import PCA

# Example data: 100 samples, 5 features
X = np.random.randn(100, 5)

# Initialize PCA to keep 2 principal components
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

print("Original shape:", X.shape)
print("Transformed shape:", X_pca.shape)
print("Explained variance ratio:", pca.explained_variance_ratio_)
```

2. numPy

```python
# Center data
X_centered = X - np.mean(X, axis=0)

# Compute covariance matrix
C = np.cov(X_centered, rowvar=False)

# Eigen decomposition
eigvals, eigvecs = np.linalg.eigh(C)

# Sort eigenvalues descending
idx = np.argsort(eigvals)[::-1]
eigvecs = eigvecs[:, idx]
eigvals = eigvals[idx]

# Project data onto first 2 components
X_pca_manual = X_centered @ eigvecs[:, :2]
print("Shape of PCA projection:", X_pca_manual.shape)
```

The linear way – PCA

Other variants of PCA:

- Probabilistic PCA (Tipping & Bishop, J. R. Statist. Soc. B 61, 611–622, 1999)
- Kernel PCA (Schölkopf *et al.*, ICANN'97, Springer, Berlin, 1997)
- Independent Component Analysis (Hérault & Jutten, Cognitiva 85(2), 593–597, 1985)
- Non-negative Matrix Factorization (Lawton & Sylvestre, Technometrics 13(3), 617–633, 1971; Lee & Seung, Nature 401, 788–791, 1999)
- …

The linear way – PCA

Other variants of PCA:

- Probabilistic PCA (Tipping & Bishop, J. R. Statist. Soc. B 61, 611–622, 1999)
- Kernel PCA (Schölkopf *et al.*, ICANN'97, Springer, Berlin, 1997)
- Independent Component Analysis (Hérault & Jutten, Cognitiva 85(2), 593–597, 1985)
- Non-negative Matrix Factorization (Lawton & Sylvestre, Technometrics 13(3), 617–633, 1971; Lee & Seung, Nature 401, 788–791, 1999)
- …

Any questions at this point?

10-min break

Outline

Lecture 1. Representation-learning methods

09:00 – 09:40: Unsupervised learning and linear approaches

09:50 – 10:30: Non-linear approaches and sparse coding

Lecture 2. Clustering methods

10:50 – 11:30: Centroid-based techniques

11:40 – 12:20: Mixture models

Beyond the linear way

What happens when features are no longer linearly related?

Beyond the linear way

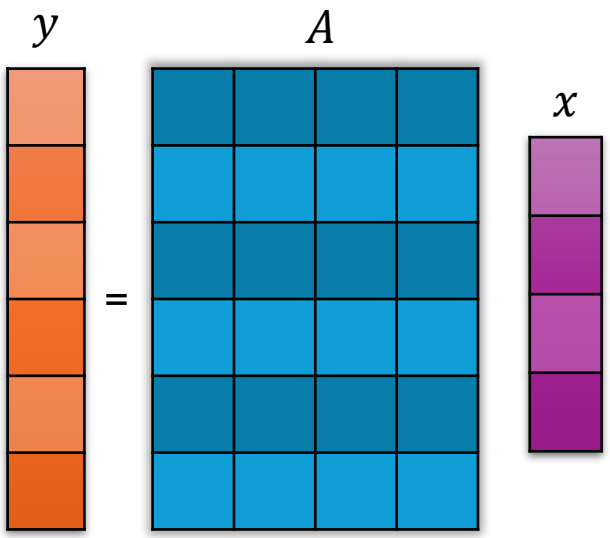
Linear mappings

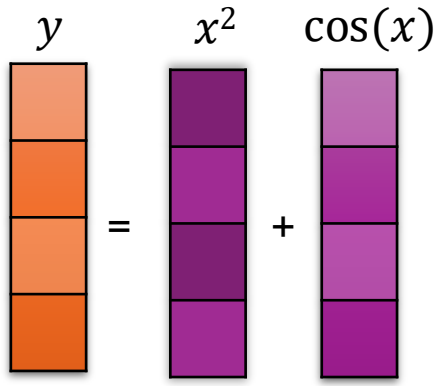
$y = A \cdot x$

Vs.

Non-linear mappings

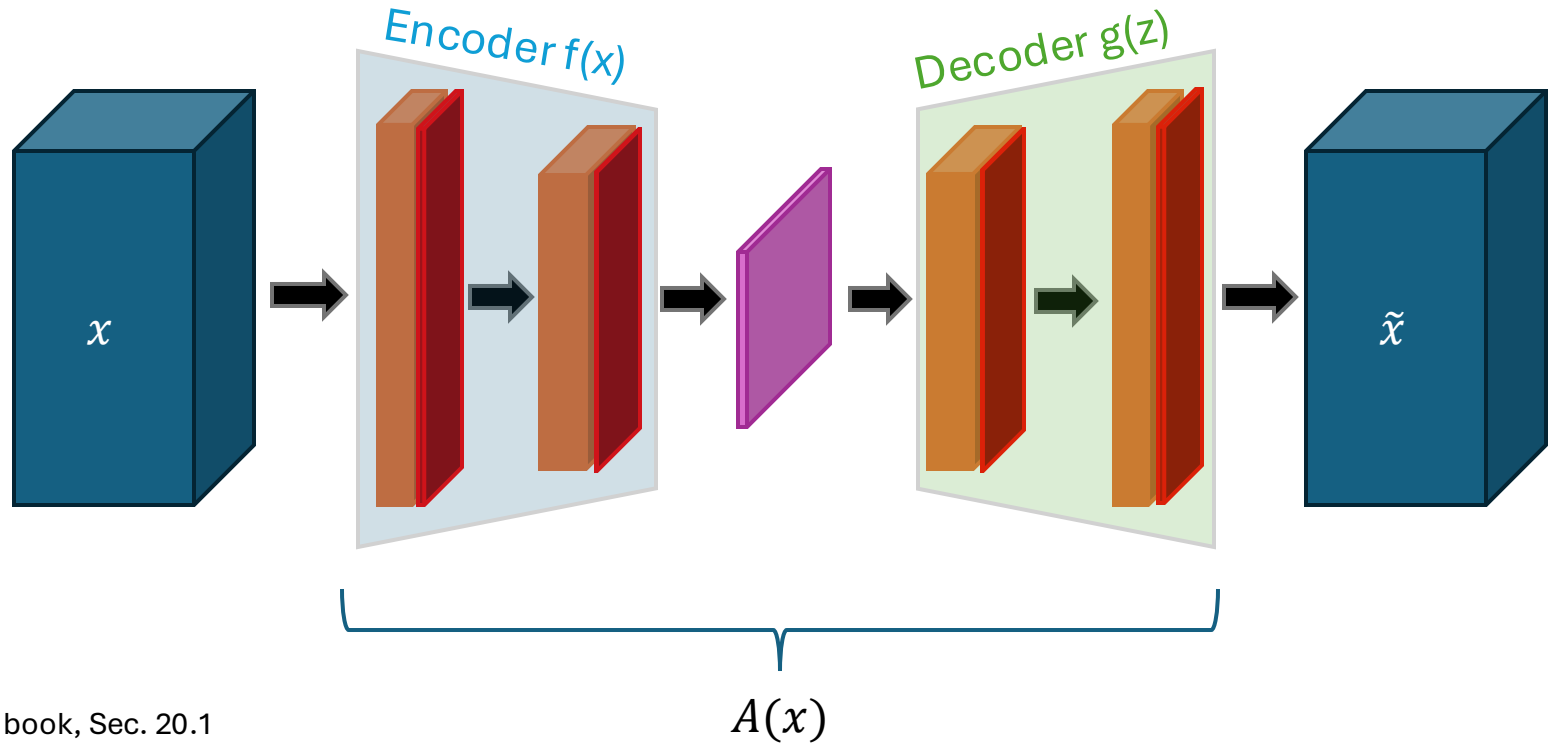
$y = A(x)$





The non-linear way – AEs

Goal: Learn the input through a constrained latent space layer



Mapping via:

$$A(x) = g(f(x))$$

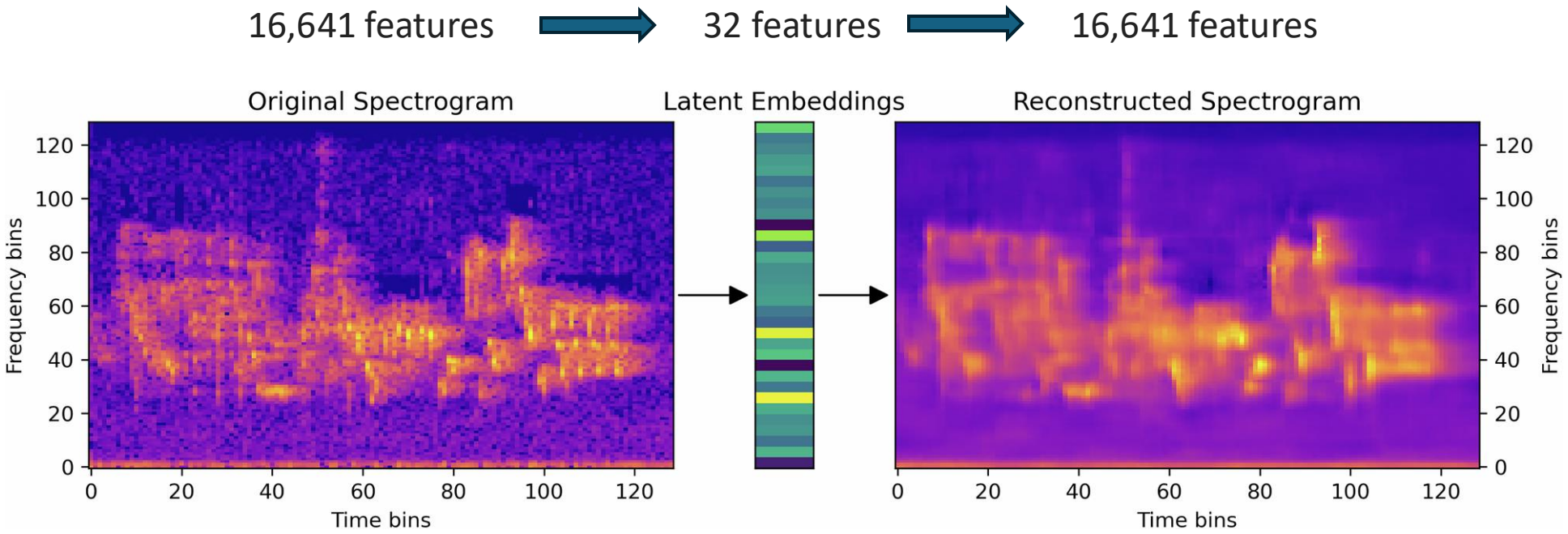
Hidden layer (FC, Conv…)

Activation function

Latent space

Ref: Murphy's book, Sec. 20.1

The non-linear way – AEs

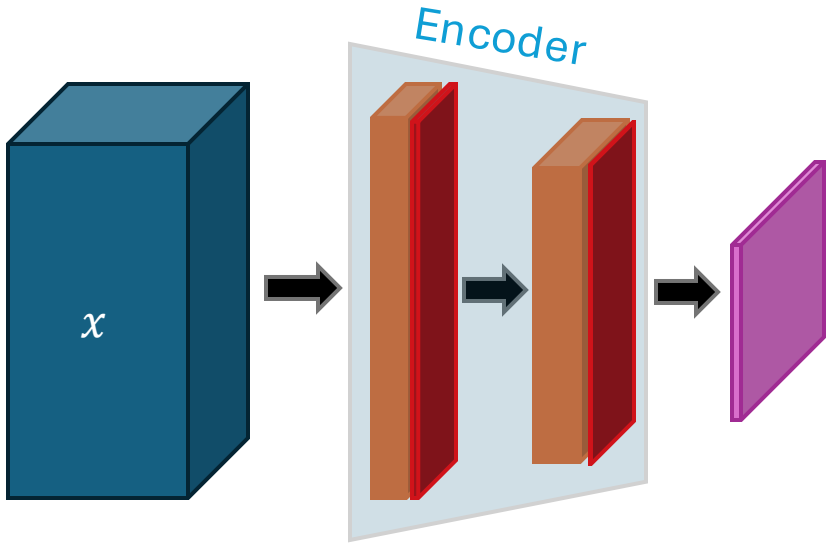


A spectrogram (left) is provided to a convolutional autoencoder trained on birdsong. The 32-dimensional latent embeddings (center) contain the salient features required to produce the reconstructed spectrogram (right).

Ref: McCarthy *et al.*, npj Acoustics 1, 18 (2025)

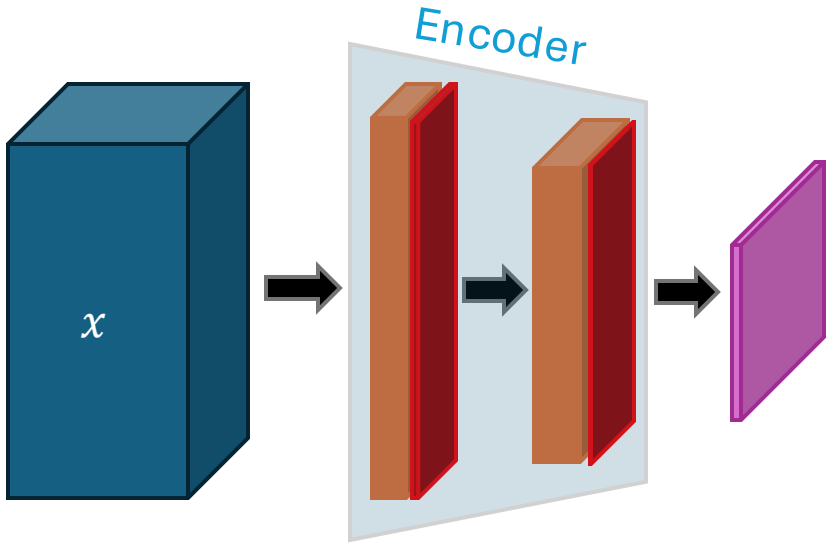
The non-linear way – AEs

Latent space (bottleneck): Lower-dimensional space compressing non-linear mapping



The non-linear way – AEs

Latent space (bottleneck): Lower-dimensional space compressing non-linear mapping



Characteristics

- Typically non-orthogonal (“entangled”)
- Encoding linearity comes from the choice of activation functions
  - Linear AE = PCA
- Can be used as a generative method (*more on Thursday!*)

The non-linear way – AEs

When are they useful?

Most effective when data lie on a complicated non-linear manifold that linear methods cannot unfold.

Since AEs can incorporate domain-specific architectures (e.g., convolutional layers for spectrograms or recurrent layers for sequential audio), they can capture structure like temporal dynamics or local spectral patterns that PCA would miss.

Computational complexity: Dictated by training / backprop complexity of the NNs

AEs – Frameworks

Some relevant frameworks:

1. PyTorch

```python
import torch
import torch.nn as nn
import torch.optim as optim

# Define Autoencoder
class Autoencoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.encoder = nn.Sequential(
            nn.Linear(784, 128),
            nn.ReLU(),
            nn.Linear(128, 32))
        self.decoder = nn.Sequential(
            nn.Linear(32, 128),
            nn.ReLU(),
            nn.Linear(128, 784),
            nn.Sigmoid())

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x

# Initialize model, loss, optimizer
model = Autoencoder()
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)

# Dummy training loop
for epoch in range(10):
    x = torch.randn(64, 784)  # batch of flattened images
    output = model(x)
    loss = criterion(output, x)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    print(f'Epoch {epoch+1}, Loss: {loss.item():.4f}')
```

2. Keras

```python
from tensorflow.keras import layers, models, optimizers
import numpy as np

# Define Autoencoder
input_dim = 784
input_img = layers.Input(shape=(input_dim,))
encoded = layers.Dense(128, activation='relu')(input_img)
encoded = layers.Dense(32, activation='relu')(encoded)
decoded = layers.Dense(128, activation='relu')(encoded)
decoded = layers.Dense(784, activation='sigmoid')(decoded)

autoencoder = models.Model(input_img, decoded)
autoencoder.compile(optimizer=optimizers.Adam(1e-3), loss='mse')

# Dummy data
x_train = np.random.rand(1000, 784)
autoencoder.fit(x_train, x_train, epochs=10, batch_size=64)
```

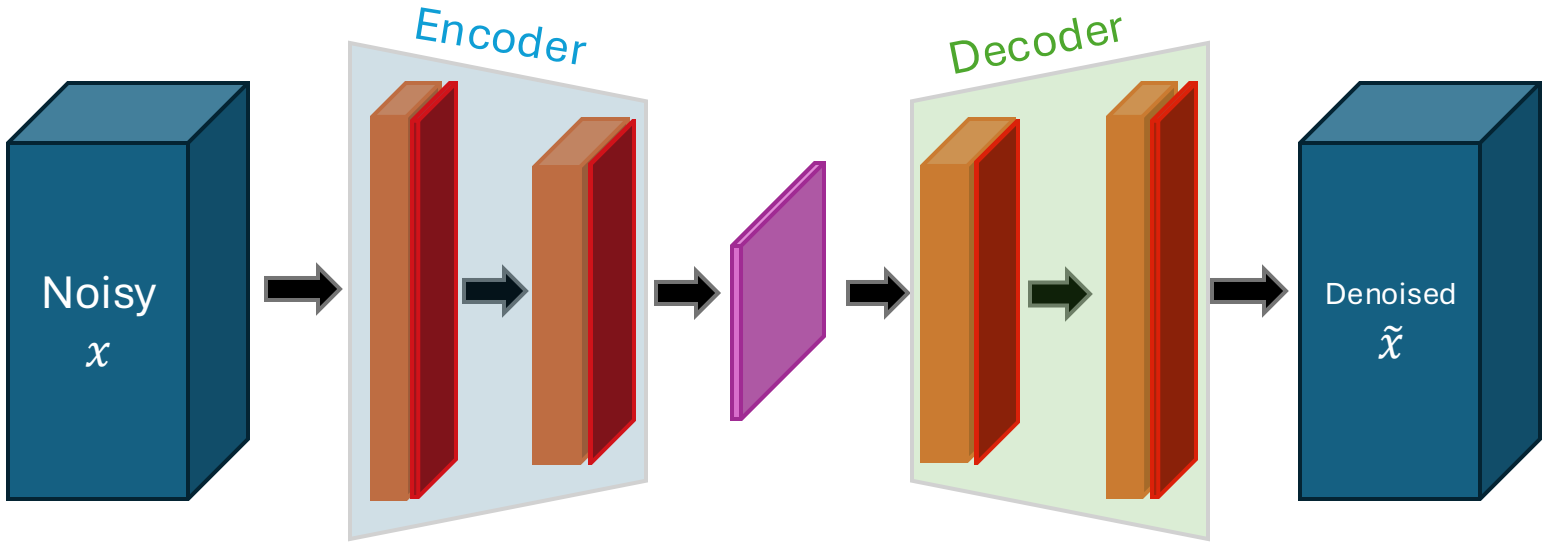
3. auDeep (Freitag *et al.*, 2017)

```python
import numpy as np
from audeep import seq2seq_autoencoder
from audeep import extract_features

# 1. Prepare your time-series (e.g., audio, vibration) data: shape (n_samples, time_steps, n_features
X = np.random.randn(100, 200, 1)  # example dummy data: 100 sequences of length 200, 1 channel

# 2. Define and train a sequence-to-sequence autoencoder
ae = seq2seq_autoencoder.SequenceToSequenceAutoencoder(
    input_shape=(200,1),
    hidden_size=64,
    num_layers=2,
    dropout=0.2
)

ae.fit(X, X, batch_size=16, epochs=30, validation_split=0.1)

# 3. Encode to get feature representations
Z = ae.encode(X)  # shape (100, hidden_size)

# 4. Alternatively, extract features using audeep's high-level API
features = extract_features.extract_from_sequences(
    sequences=X,
    autoencoder=ae
)

print("Encoded feature shape:", Z.shape)
print("Extracted features shape:", features.shape)
```

Applications of AEs

Signal denoising & anomaly detection

Denoising with AEs



"Fool" the AE until it learns to clean any input ☺

Refs:

- Zhou *et al.*, J. Phys.: Conf. Ser. 2031 012001 (2021)
- Dong *et al.*, IEEE Trans. Instrum. Meas. 71, 1–8 (2022)
- Mao *et al.*, Neurocomputing 625, 129607 (2025)

Denoising with AEs

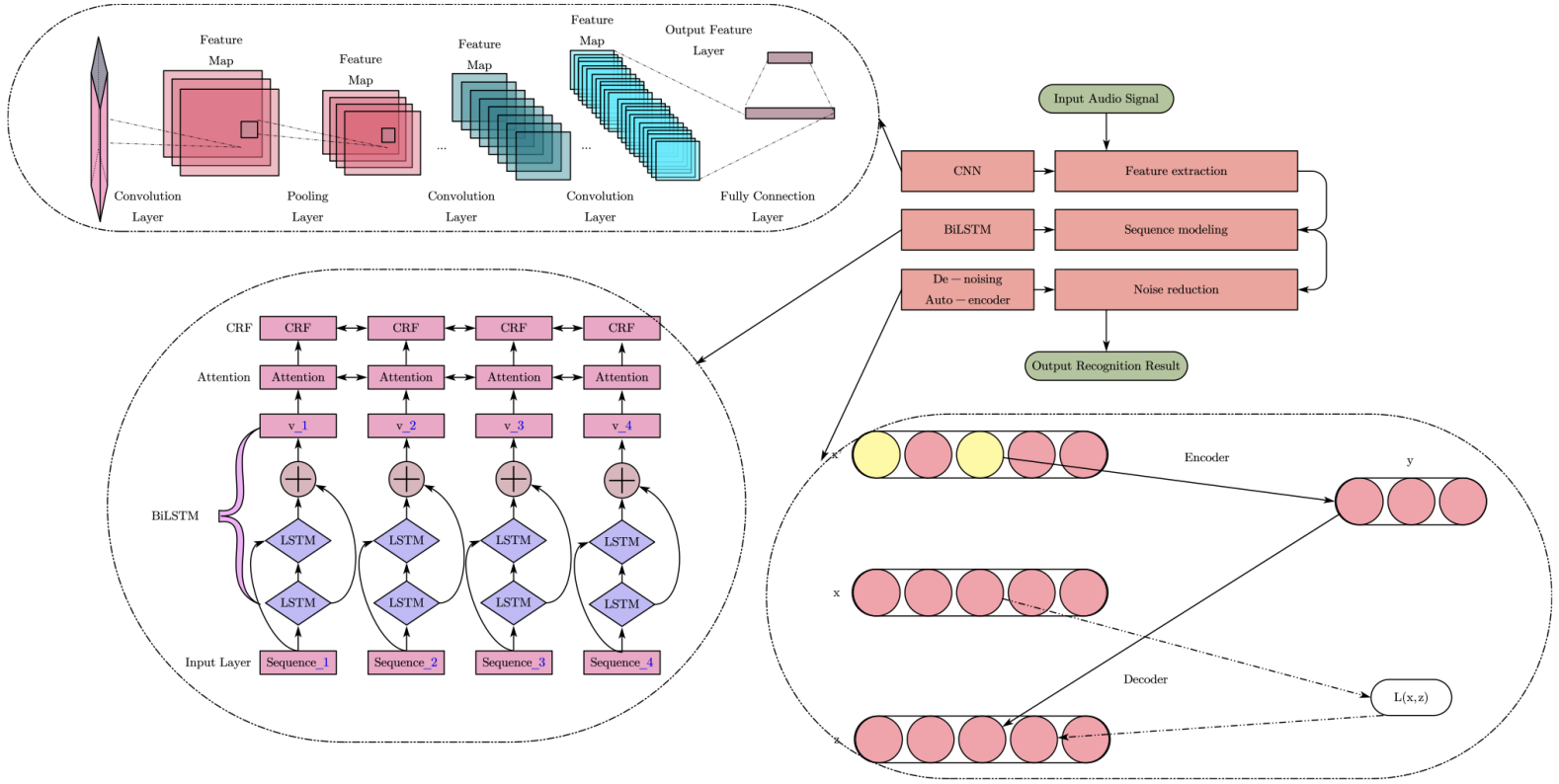
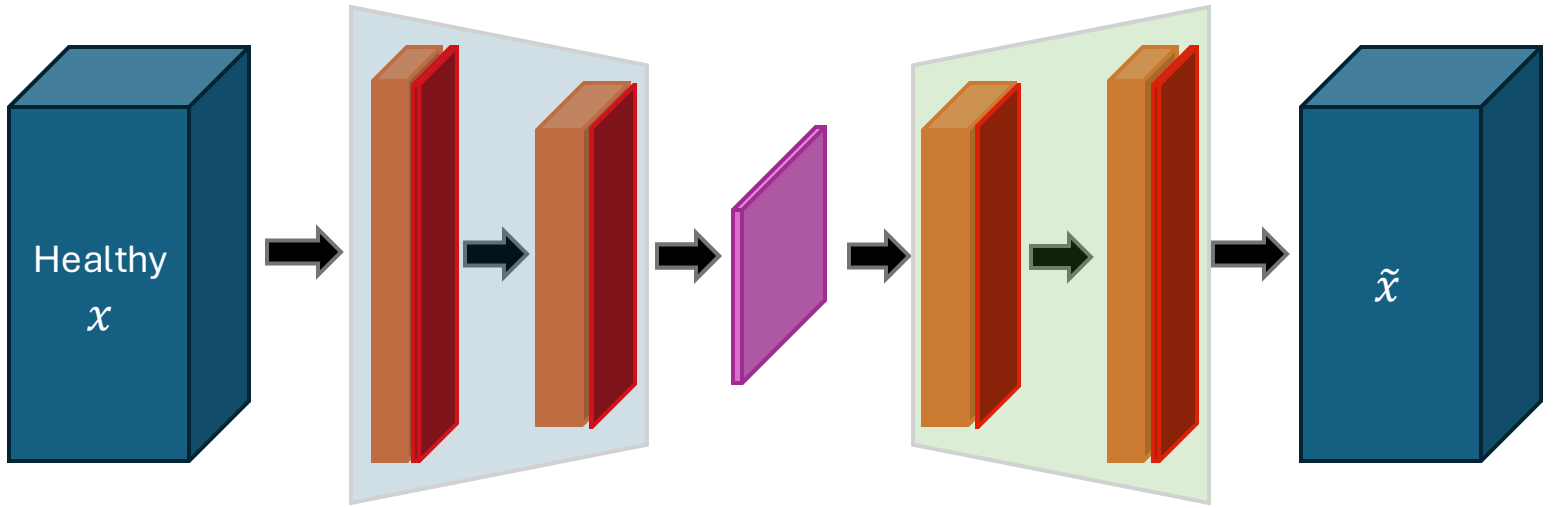


Fig. 14. The De-noising auto-encoder structure.

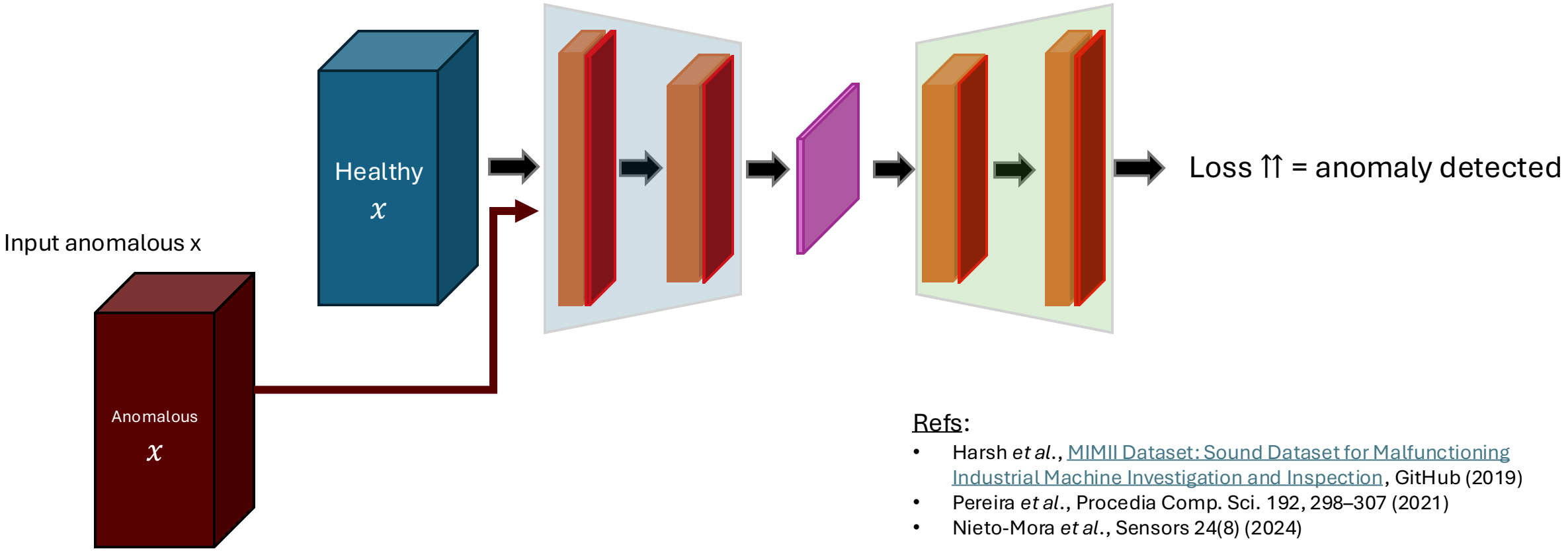
Mao *et al.*, Neurocomputing 625, 129607 (2025)

Anomaly detection with AEs

Training stage with non-anomalous cases



Anomaly detection with AEs



Refs:

- Harsh *et al.*, MIMII Dataset: Sound Dataset for Malfunctioning Industrial Machine Investigation and Inspection, GitHub (2019)
- Pereira *et al.*, Procedia Comp. Sci. 192, 298–307 (2021)
- Nieto-Mora *et al.*, Sensors 24(8) (2024)

Anomaly detection with AEs

Application: In-vehicle audio anomaly detection

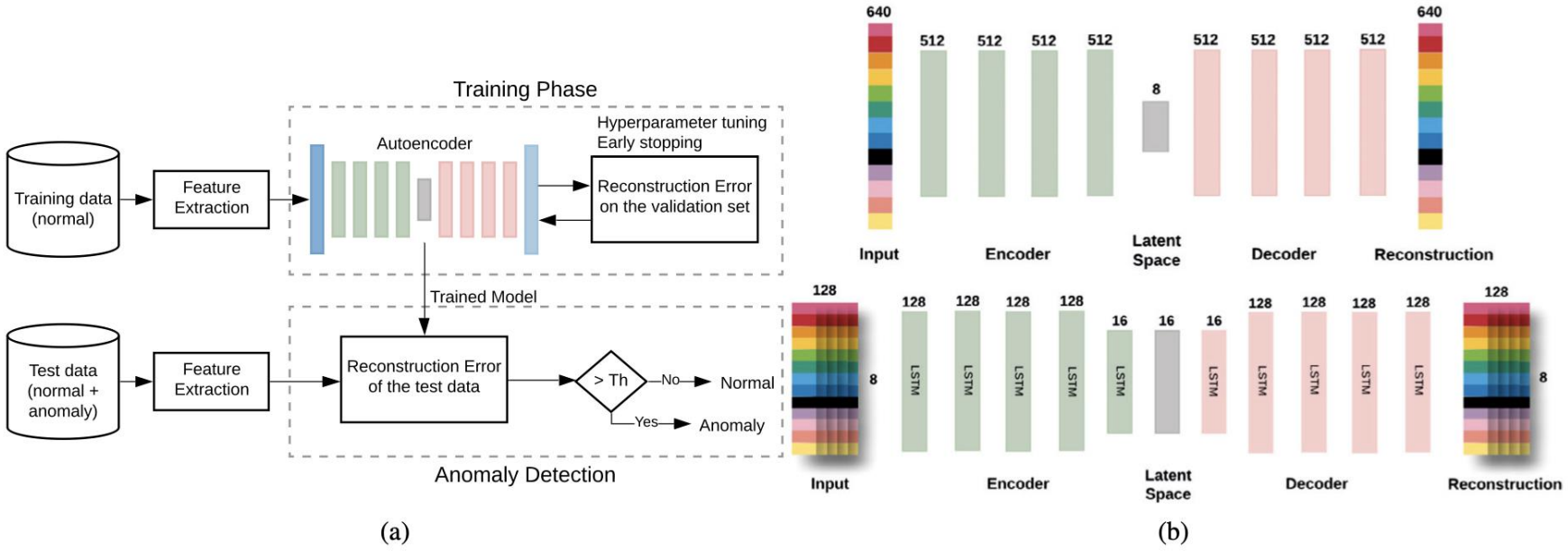


Fig. 4: Schematic of the AE AAD system (a) and the two AE architectures (b): Dense-AE (top) and LSTM-AE (bottom).

Pereira *et al.*, Procedia Comp. Sci. 192, 298–307 (2021)

Now let us move to sparse coding…

…but first, what is sparse optimization?

What is sparse optimization?

Some geometrical intuition…

In mathematical terms:

$$\min_{\mathbf{s}} \|\mathbf{s}\|_1 \quad \text{subject to} \quad \tilde{\mathbf{y}} = \mathbf{\Phi s}$$

Given a linear synthesis operator $\mathbf{\Phi}$, what is the vector $\mathbf{s}$ that, when multiplied with $\mathbf{\Phi}$, has the smallest ℓ_1 norm while fulfilling the linear equation $\tilde{\mathbf{y}} = \mathbf{\Phi s}$?

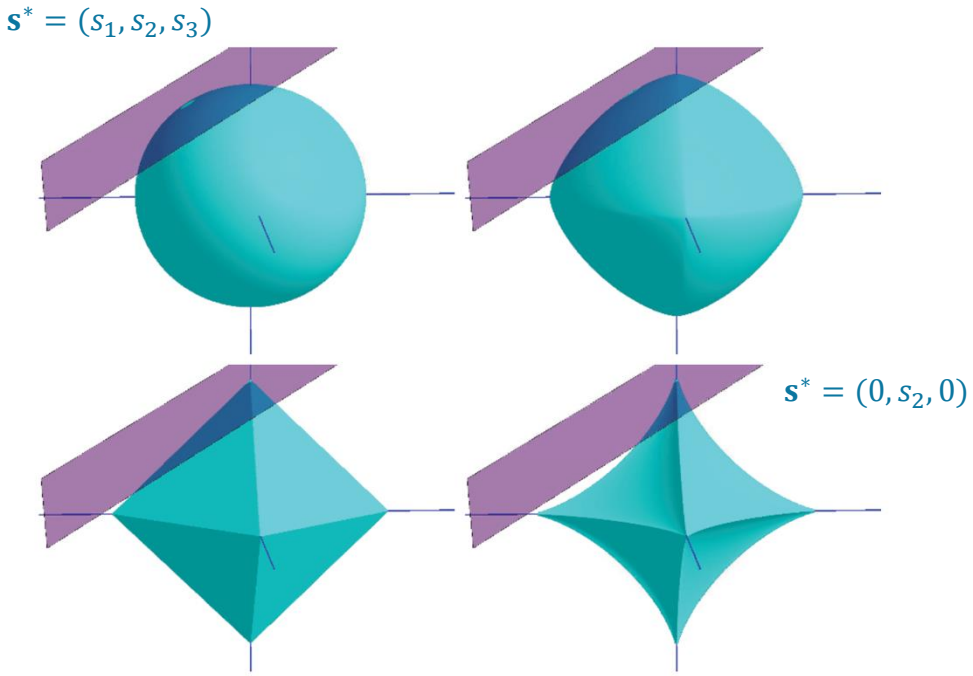
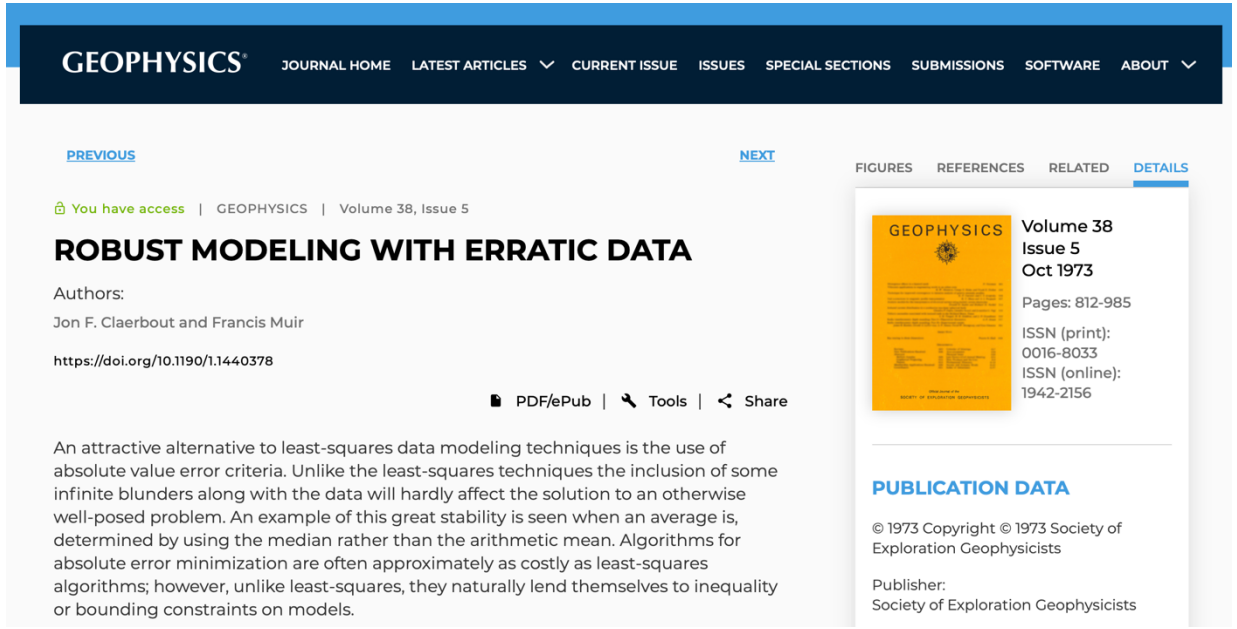


Fig. 1.2 The intersection between the ℓ_p-ball and the set $\mathbf{Ax} = \mathbf{b}$ defines the solution of (P_p). This intersection is demonstrated here in 3D for $p = 2$ (top left), $p = 1.5$ (top right), $p = 1$ (bottom left), and $p = 0.7$ (bottom right). When $p \leq 1$, the intersection takes place at a corner of the ball, leading to a sparse solution.

Elad, Sparse and Redundant Representations, Springer-Verlag (2010)

A brief history of sparse coding

The rise of sparse optimization: Claerbout and Muir on erratic seismic data…



Main idea:
Summing absolute errors instead of squared errors

$$m_2 := m \Big| \sum_{i=1}^{N} (m - x_i)^2 \quad \text{is min.}$$

$$m_1 := m \Big| \sum_{i=1}^{N} |m - x_i| \quad \text{is min.}$$

A brief history of sparse coding

It’s 1996 and neuroscientists are trying to understand receptive fields in our visual system…

nature

Explore content ⌄ About the journal ⌄ Publish with us ⌄

nature › letters › article

Letter | Published: 13 June 1996

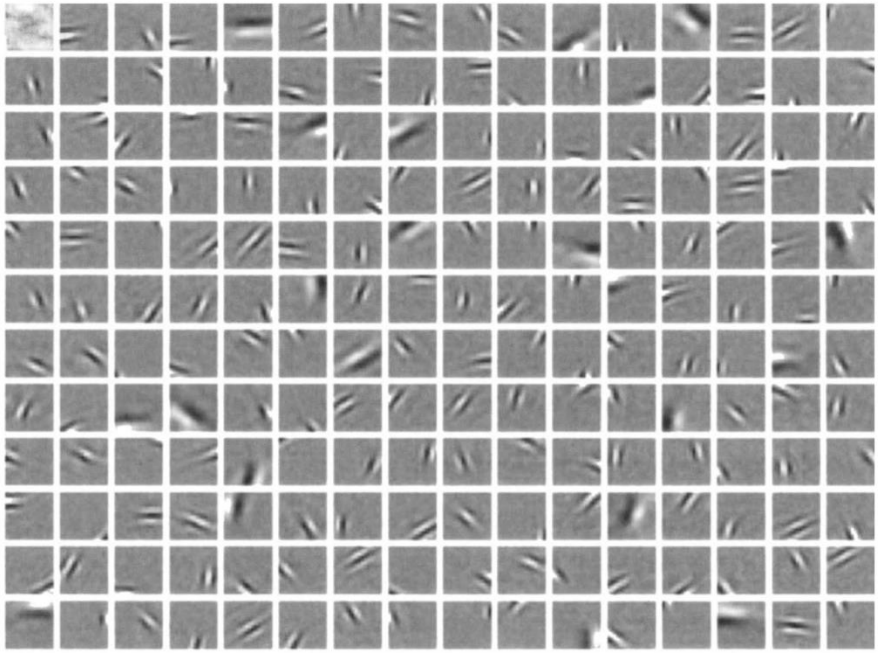
Emergence of simple-cell receptive field properties by learning a sparse code for natural images

Bruno A. Olshausen & David J. Field

Nature **381**, 607–609 (1996) | Cite this article

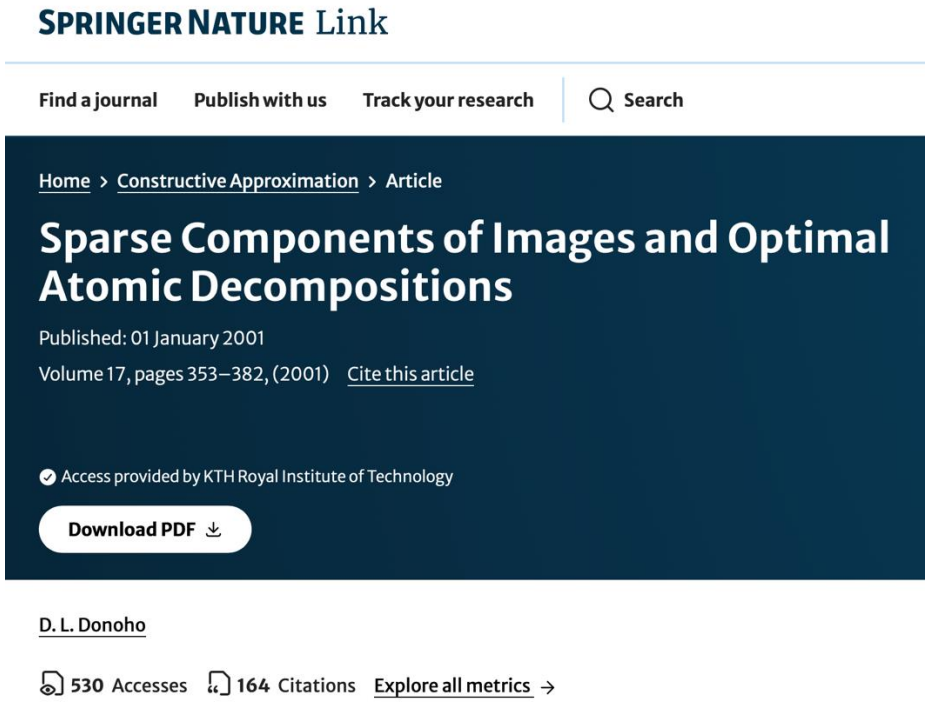
31k Accesses | **4603** Citations | **73** Altmetric | Metrics

Sparse PCA results



A brief history of sparse coding

Few years later, mathematicians started looking at handcrafting analytical sparse codes of images…



Change from database $\mathcal{X}$ to functional class $\mathcal{F}$. The original problem required the analysis of a database of image patches sampled from naturally occurring data. Instead, we study a mathematically defined class of objects $\mathcal{F}$. The point of this replacement is, of course, that we expect to gain a certain mathematical structure which allows us to say something about the continuum situation.

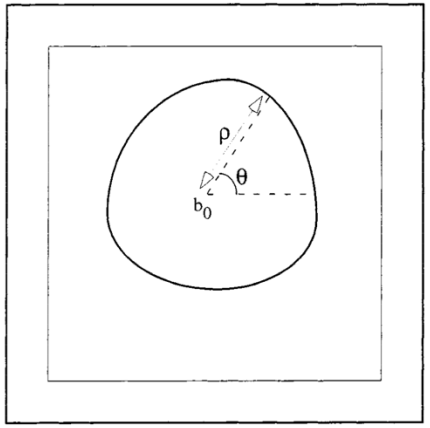


Fig. 2. Typical star-shaped set, and associated notation.

Given a class of star-shaped functions, what is the optimally sparse representation?

A brief history of sparse coding

This led to the development of analytical sparse codes for various classes of functions:

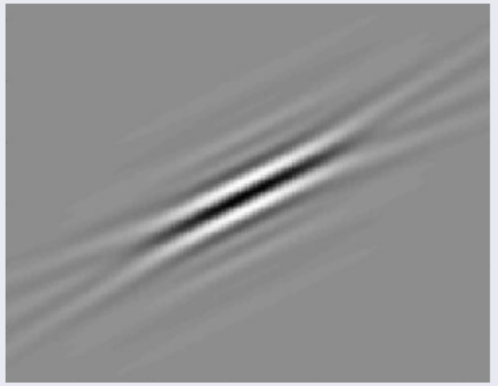
Ridgelets

Line discontinuities

Candès & Donoho, Phil. Trans.: Math. Phys. Eng. Sci. 357, 2495–2509 (1999)

Curvelets/shearlets

Curve discontinuities



Candès & Donoho, Comm. Pure Appl. Math. 57(2), 219–266 (2004)

Dictionary learning

What is a dictionary?

Dictionary learning

What is a dictionary?

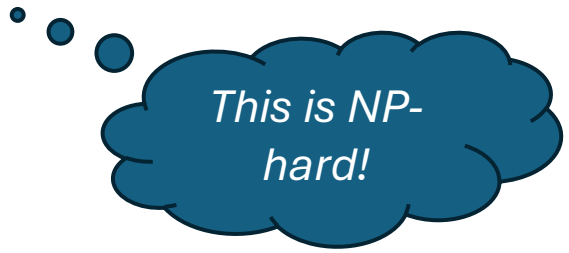
Can you think of an example?

Dictionary learning – *K*-SVD

Goal: Learn a (usually non-orthogonal) collection of atoms $D \in \mathbb{R}^{N \times K}$ that sparsely represent the variance of the data

Some algorithms include *K*-SVD and sparse PCA. For example, *K*-SVD solves the optimization problem

$$\hat{\mathbf{A}}, \hat{\mathbf{D}} = \arg\min_{\mathbf{D}} \left\{ \begin{array}{ll} \arg\min_{\mathbf{a}_m} \|\mathbf{D}\mathbf{a}_m - \mathbf{x}_m\|^2 & \\ \text{subject to} & \|\mathbf{a}_m\|_0 = T \; \forall m \end{array} \right\}$$



where $\|\mathbf{a}_m\|_0$ is the so-called ℓ_0 pseudonorm of $\mathbf{a}_m$.

Aharon, Elad & Bruckstein, IEEE Trans. Signal Process. 54(11), 4311–4322 (2006).

Dictionary learning – Frameworks

Some relevant frameworks

1. pyKSVD

```python
import numpy as np
from pyksvd.pyksvd import KSVD

# 1) Prepare data: extract patches from your 2D data
#    Example: assume `images` is an array of shape (n_images, H, W)
n_images = 100
H, W = 32, 32
images = np.random.randn(n_images, H, W)

# Flatten patches (e.g., extract non-overlapping 8×8 patches)
patch_h, patch_w = 8, 8
patches = []
for img in images:
    for i in range(0, H, patch_h):
        for j in range(0, W, patch_w):
            p = img[i:i+patch_h, j:j+patch_w]
            patches.append(p.flatten())
X = np.stack(patches, axis=1)  # shape: (patch_h*patch_w, n_patches)
print("X shape:", X.shape)

# 2) Fit K-SVD
K = 100      # number of dictionary atoms
T0 = 5       # sparsity (max non-zero coefficients per signal)
ksvd = KSVD(K=K, T0=T0)
ksvd.fit(X, verbose=True)

# 3) Retrieve dictionary D and sparse codes X_rec, etc.
D = ksvd.D      # shape: (patch_h*patch_w, K)
X_codes = ksvd.X   # shape: (K, n_patches)
print("Dictionary shape:", D.shape)
print("Codes shape:", X_codes.shape)
```

2. scikit-learn (K-SVD)

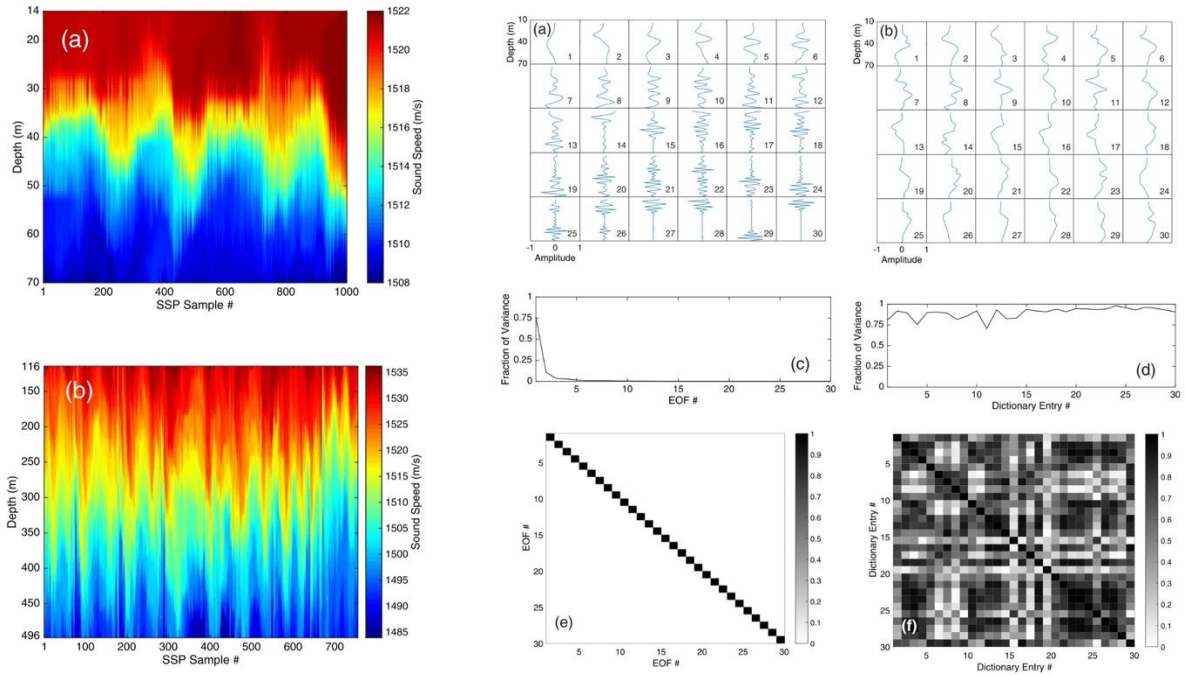
```python
from sklearn.decomposition import DictionaryLearning

X = np.random.randn(64, 1000)
dict_learner = DictionaryLearning(n_components=100, alpha=0.15, max_iter=20)
A = dict_learner.fit_transform(X.T)
D = dict_learner.components_.T
print("Dictionary shape:", D.shape)
```

3. scikit-learn (SPCA)

```python
import numpy as np
from sklearn.decomposition import SparsePCA
import matplotlib.pyplot as plt

# 1) Prepare data: 2D patches (e.g., from images or space–time frames)
n_images = 100
H, W = 32, 32
images = np.random.randn(n_images, H, W)

# Extract non-overlapping 8×8 patches and flatten them
patch_h, patch_w = 8, 8
patches = []
for img in images:
    for i in range(0, H, patch_h):
        for j in range(0, W, patch_w):
            p = img[i:i+patch_h, j:j+patch_w]
            patches.append(p.flatten())
X = np.stack(patches, axis=0)  # shape: (n_patches, patch_h*patch_w)
print("X shape:", X.shape)

# 2) Fit Sparse PCA
K = 100  # number of components (atoms)
alpha = 1.0  # sparsity control (higher → sparser)
spca = SparsePCA(n_components=K, alpha=alpha, max_iter=100, random_state=0)
codes = spca.fit_transform(X)   # shape: (n_patches, K)
D = spca.components_.T          # shape: (patch_h*patch_w, K)
```

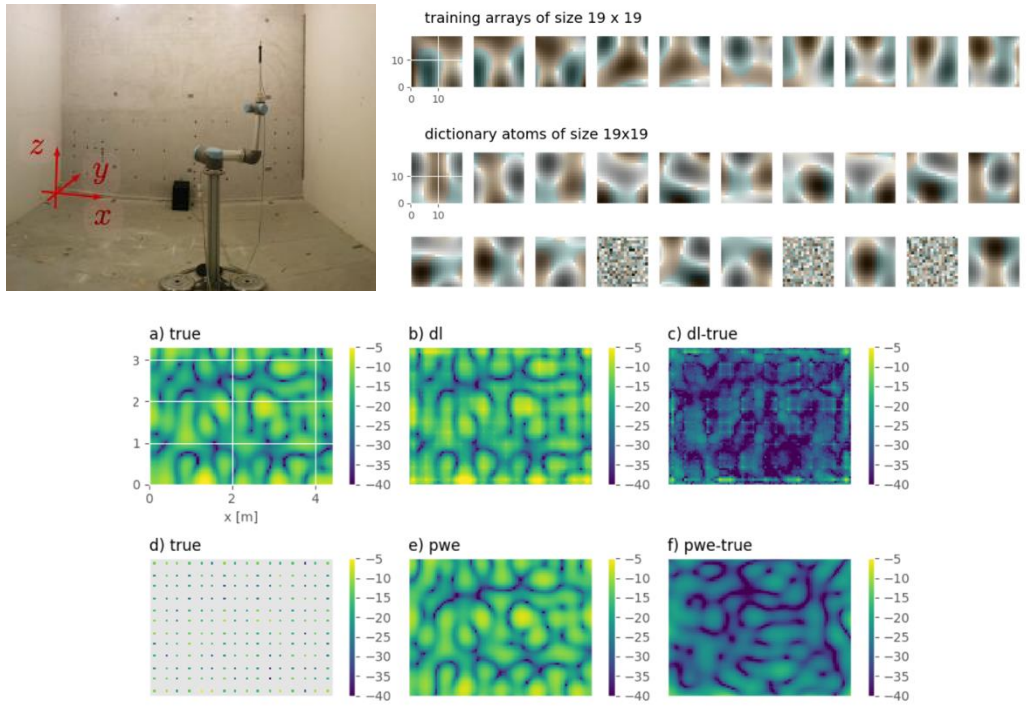
Applications of sparse coding

Ex. Learning ocean sound speed profiles from experiments



Bianco & Gerstoft, JASA 141(3), 1749–1759 (2017)

Ex. Dictionary learning of the sound field in a room

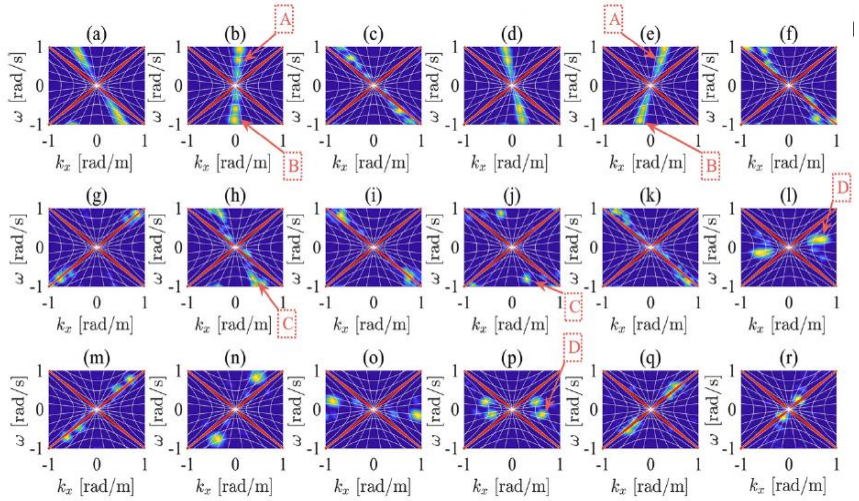


Hahmann *et al.*, Proc. ICA2019, Aachen, 149–154 (2019)

Applications of sparse coding

1. Unsupervised learning

Sparse coding



Zea & Laudato, JASA Exp. Lett 1(6), 0004852 (2021)

2. Design of basis functions

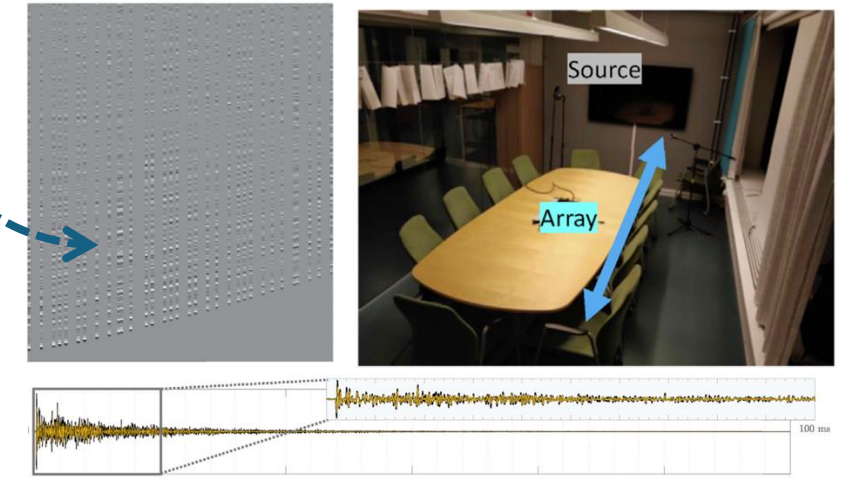
Handcraft functions according to the sparse codes

$$\|f\|^2 = \int_{\mathbb{R}^2} |\langle f, \phi_\ell \rangle|^2 \mathrm{d}\ell + \int_{\mathbb{B}} |\langle f, \psi_{a,\theta,\ell} \rangle|^2 \frac{\mathrm{d}a\mathrm{d}\theta\mathrm{d}\ell}{a^3} + \int_{\mathbb{B}} |\langle f, \tilde{\psi}_{a,\theta,\ell} \rangle|^2 \frac{\mathrm{d}a\mathrm{d}\theta\mathrm{d}\ell}{a^3}$$
$$= \int_{\mathbb{R}^2} |\hat{f}(\omega)|^2 |\hat{\phi}(\omega)|^2 \mathrm{d}\omega + \int_{\mathbb{R}^2} |\hat{f}(\omega)|^2 \mathrm{d}\omega \iint_{\mathcal{S}_1} \frac{|\hat{\psi}(\xi_x, \xi_y)|^2}{\xi_y^2 - \xi_x^2} \mathrm{d}\xi_x \mathrm{d}\xi_y$$
$$+ \int_{\mathbb{R}^2} |\hat{f}(\omega)|^2 \mathrm{d}\omega \iint_{\mathcal{S}_2} \frac{|\hat{\tilde{\psi}}(\xi_x, \xi_y)|^2}{\xi_x^2 - \xi_y^2} \mathrm{d}\xi_x \mathrm{d}\xi_y,$$

Zea & Laudato, JASA Exp. Lett 1(6), 0004852 (2021)

3. Sparse sampling

Measure waves with fewer microphones



AEs vs. Sparse coding

Key message:

Both autoencoders (AEs) and sparse coding aim to find compact, meaningful representations of data — the main difference lies in **how** the representation is parameterized and enforced.

	AEs	Sparse coding
Representation structure	Encoder $z = f_\theta(x)$ and decoder $\tilde{x} = g_\phi(z)$, given learnable ϕ and θ	Dictionary & suitable weights, $\tilde{x} \approx Ds$, given learnable *D* and *s*
Sparsity mechanism	Implicit regularizer in the architecture (bottleneck, activations, dropout, etc.)	Explicit regularizer in the code
Objective function	$\min_{\theta,\phi} \|x - g_\phi(f_\theta(x))\|^2 + R(z)$	$\min_{\theta,\phi} \|x - Ds\|^2 + \mu\|s\|_1$
Interpretability	Latent variables z span a non-linear manifold	Atoms in *D* correspond to “basis functions”

Lecture 1 – Summary and Q&A

- Representation learning provides a way to learn (*and handcraft*) interpretable, efficient data representations.
- We covered algorithms from linear (e.g., PCA) to non-linear and more advanced (e.g., K-SVD) that learn representative features / atoms.
- **Rule of thumb:** When approaching a new problem, consider first if the data might have a sparse structure. If yes, exploring dictionary learning or enforcing sparsity in a model can yield benefits like faster inference (only a few components active) and possibly better generalization (by ignoring irrelevant details).
- We also discussed several applications of representation learning and sparse coding in acoustics.

Questions?

20-min break

Outline

Lecture 1. Representation-learning methods

09:00 – 09:40: Unsupervised learning and linear approaches
09:50 – 10:30: Non-linear approaches and sparse coding

Lecture 2. Clustering methods

10:50 – 11:30: Centroid-based techniques
11:40 – 12:20: Mixture models

Lecture 2.1. Centroid-based methods

- K-means

10-min break

Outline

Lecture 1. Representation-learning methods

09:00 – 09:40: Unsupervised learning and linear approaches
09:50 – 10:30: Non-linear approaches and sparse coding

Lecture 2. Clustering methods

10:50 – 11:30: Centroid-based techniques
11:40 – 12:20: Mixture models

Lecture 2.2. Mixture models

- GMMs